WHAISPER

Memoria

Autores:

Cesar Arenas, Adonay Rivas

Última actualización:

23/05/2025

Índice

MEMORIA DEL PROYECTO

Bot de Telegram para Gestión de Inventario y Ventas

Autores: Cesar Arenas, Adonay Rivas
Fecha: 21/05/2025
Versión: 1.0.0.1

1. Elementos Preliminares

1.1. Resumen Ejecutivo

El presente proyecto implementa una solución de comercio electrónico a través de Telegram, permitiendo a pequeñas empresas gestionar su inventario y ventas de manera sencilla mediante un bot conversacional. La aplicación utiliza tecnologías modernas como Node.js, MongoDB, y APIs de OpenAI para ofrecer una experiencia de usuario natural y eficiente.

Objetivos principales:

- Desarrollar un bot de Telegram capaz de gestionar inventario y procesar pedidos
- Implementar un sistema de búsqueda inteligente basado en embeddings vectoriales
- Crear un carrito de compras funcional con gestión de stock en tiempo real
- Proporcionar una interfaz de administración para subir y gestionar productos
- Utilizar procesamiento de lenguaje natural para entender las intenciones del usuario

Resultados clave conseguidos:

- Sistema completo con capacidad de procesamiento de lenguaje natural y voz
- Gestión inteligente del estado de conversación para una experiencia fluida
- Estructura modular y escalable para futuras ampliaciones
- Implementación de medidas de seguridad para la administración
- Alta precisión en la comprensión de intenciones del usuario

1.2. Índice

1. **Elementos Preliminares**
    - Resumen Ejecutivo
    - Índice
2. **Cuerpo Principal**
    - Introducción
    - Planificación
    - Desarrollo del Proyecto
    - Evaluación y Resultados
3. **Elementos Finales**
    - Conclusiones
    - Referencias y Bibliografía
    - Anexos

2. Cuerpo Principal

2.1. Introducción

Contexto del proyecto

En la actualidad, los pequeños negocios necesitan soluciones tecnológicas accesibles para competir en el mercado digital. Telegram, con más de 700 millones de usuarios activos, representa una plataforma ideal para implementar soluciones de comercio electrónico sin la necesidad de desarrollar aplicaciones móviles dedicadas. Este proyecto nace de la necesidad de ofrecer una alternativa eficiente y de bajo coste para que pequeños comercios puedan gestionar su inventario y ofrecer sus productos a través de una plataforma de mensajería ampliamente utilizada.

Objetivos detallados

1. **Diseñar un sistema conversacional intuitivo:**
    - Implementar un análisis de intención mediante IA para comprender las peticiones de los usuarios
    - Crear un sistema de estados de conversación para mantener el contexto
    - Permitir interacción mediante texto y voz
2. **Implementar gestión de inventario:**
    - Diseñar un modelo de datos eficiente para productos

    - Crear funcionalidades de importación masiva desde CSV/Excel
    - Implementar búsqueda semántica mediante embeddings vectoriales
3. **Desarrollar sistema de ventas:**
    - Implementar carrito de compra con todas sus funcionalidades
    - Gestionar stock en tiempo real
    - Crear sistema de pedidos con estados
4. **Implementar panel de administración:**
    - Diseñar sistema de permisos para administradores
    - Crear funcionalidades de gestión de inventario
    - Implementar estadísticas y reportes
5. **Garantizar robustez:**
    - Implementar manejo de errores consistente
    - Diseñar sistema de reintentos para operaciones críticas
    - Crear sistema de logging para facilitar depuración

Alcance del proyecto

El proyecto abarca el desarrollo de un sistema completo de comercio electrónico basado en Telegram, incluyendo:

- API RESTful para gestión de datos
- Bot de Telegram con capacidades conversacionales
- Sistema de administración dentro del mismo bot
- Integración con MongoDB Atlas como base de datos
- Utilización de APIs de OpenAI para procesamiento de lenguaje natural
- Sistema de transcripción de mensajes de voz

No se incluye en el alcance:

- Pasarela de pagos (se deja para futuras versiones)
- Aplicación web administrativa (se gestiona todo desde Telegram)
- Sincronización con otros sistemas de inventario

Metodología utilizada

Para el desarrollo del proyecto se ha seguido una metodología ágil basada en Scrum, con sprints de dos semanas. Las principales características de la metodología aplicada son:

- **Desarrollo iterativo**: Construcción incremental de funcionalidades
- **Reuniones diarias**: Breves stand-ups para revisar progreso y bloqueos

- **Revisiones de código**: Implementación de pull requests y code reviews
- **Integración continua**: Uso de GitHub para mantener un flujo de trabajo organizado
- **Pruebas continuas**: Testing manual de cada funcionalidad desarrollada

2.2. Planificación

Plan de trabajo general

El proyecto se ha estructurado en cuatro fases principales, con una duración total de 12 semanas:

1. **Fase de diseño y planificación** (1 semana)
    - Análisis de requisitos
    - Diseño de la arquitectura
    - Selección de tecnologías
    - Configuración del entorno de desarrollo
2. **Fase de desarrollo del backend** (1 semanas)
    - Implementación de modelos de datos
    - Desarrollo de la API REST
    - Integración con MongoDB
    - Implementación de servicios de IA
3. **Fase de desarrollo del bot** (1 semanas)
    - Implementación del flujo conversacional
    - Desarrollo de comandos y callbacks
    - Integración con servicios backend
    - Implementación de funcionalidades de administración
4. **Fase de pruebas y despliegue** (1semana)
    - Pruebas de integración
    - Corrección de errores
    - Optimización de rendimiento
    - Despliegue a producción

Recursos necesarios

Equipo:

- 2 desarrolladores

Recursos tecnológicos:

- Equipos de desarrollo
- Servidores para despliegue (DigitalOcean)
- Servicios cloud:
    - MongoDB Atlas
    - OpenAI API
    - GitHub
    - Jira
    - MongoDB

Recursos financieros:

- Presupuesto para APIs: $150/mes

Plan de trabajo detallado

Tareas semanales:

- Coordinación y repaso de la estructura del proyecto
- Actualizaciones diarias a GitHub
- Pull Requests a development como versión estable
- Reuniones semanales de revisión de sprint

Hitos alcanzados

1. **Semana 1**: Arquitectura definida y entorno configurado
2. **Semana 2**: Modelos de datos implementados y funcionando
3. **Semana 3**: API REST completamente funcional
4. **Semana 3**: Bot básico implementado con comandos principales
5. **Semana 4**: Funcionalidades de administración implementadas
6. **Semana 4**: Sistema desplegado y funcionando en producción

Desviaciones de la planificación

Durante el desarrollo del proyecto se han producido algunas desviaciones respecto a la planificación inicial:

- La implementación del análisis de intención mediante IA requirió más tiempo del previsto debido a la complejidad de los prompts y ajustes necesarios para obtener precisión.
- Se tuvo que reprogramar algunas tareas y acelerar el proceso de desarrollo del frontend y la puesta a producción.

2.3. Desarrollo del Proyecto

Descripción de la solución implementada

El sistema desarrollado es un bot de Telegram que permite a los usuarios:

- Buscar productos mediante lenguaje natural
- Ver detalles de productos
- Añadir productos al carrito
- Gestionar cantidades
- Finalizar pedidos

Para los administradores, el sistema ofrece:

- Gestión de inventario mediante subida de archivos CSV/Excel
- Visualización de pedidos
- Gestión de permisos de administración
- Estadísticas básicas de ventas

La solución implementa un sistema de análisis de intención que permite entender lo que el usuario desea hacer a partir de sus mensajes de texto o voz(NLU), manteniendo el contexto de la conversación.

Problemas comunes y soluciones

Error de Conexión a MongoDB

- **Síntoma**: Error "MongoNetworkError: failed to connect to server"
- **Posibles causas**:
    - IP no registrada en whitelist de MongoDB Atlas
    - Credenciales incorrectas en variables de entorno
    - Problemas de red o firewall
- **Solución**:
    - Verificar que la IP del servidor esté en la whitelist de MongoDB Atlas
    - Comprobar las credenciales en el archivo .env
    - Verificar conexión de red y configuración de firewall

Error en la Generación de Embeddings

- **Síntoma**: Error "Authentication error" al ejecutar generateEmbeddings.js
- **Posibles causas**:
    - API Key de OpenAI incorrecta o inválida
    - Cuota de API agotada
- **Solución**:
    - Verificar la API Key en el archivo .env
    - Comprobar el estado de la cuota en el dashboard de OpenAI
    - Ejecutar con la opción --verbose para obtener más detalles

Bot de Telegram No Responde

- **Síntoma**: El bot no responde a mensajes
- **Posibles causas**:
    - Token de bot incorrecto
    - Servidor no ejecutándose
- **Solución**:
    - Verificar el token en .env
    - Comprobar logs del servidor
    - Reiniciar el bot

Tecnologías utilizadas

Stack Tecnológico

- **Node.js**: Entorno de ejecución para JavaScript del lado del servidor
- **Express.js (v5.1.0)**: Framework web para la creación de APIs
- **OpenAI (v4.97.0)**: Para integración con OpenAI API (embeddings y análisis de intención)
- **Mongoose (v8.14.1)**: ODM para MongoDB
- **Node-Telegram-Bot-API (v0.66.0)**: Para la integración con Telegram
- **Dotenv (v16.5.0)**: Para gestión de variables de entorno
- **Multer (v1.4.5-lts.2)**: Para manejo de subida de archivos
- **XLSX (v0.18.5)**: Para procesamiento de archivos Excel
- **CSV-Parser (v3.2.0)**: Para procesamiento de archivos CSV
- **Axios (v1.9.0)**: Para peticiones HTTP
- **FS-Extra (v11.3.0)**: Extensión de operaciones de sistema de archivos
- **Compute-Cosine-Similarity (v1.1.0)**: Para cálculo de similitud coseno en embeddings
- **Winston (v3.10.0)**: Sistema de logging avanzado

Componentes principales

Frontend:

- **Telegram**: La interfaz de usuario es proporcionada por la aplicación de Telegram mediante mensajes interactivos con botones.

Backend:

- **Node.js**: Servidor principal que gestiona toda la lógica de la aplicación
- **Controllers**: Conjunto de módulos que procesan tipos específicos de interacciones
- **Services**: Componentes que implementan la lógica de negocio
- **Models**: Definiciones de estructura de datos para MongoDB

Base de datos:

- **MongoDB Atlas**: Almacena productos, pedidos y datos de usuarios

Servicios externos:

- **OpenAI API**: Para generación de embeddings vectoriales y análisis de intención
- **Telegram Bot API**: Para la integración del bot de Telegram
- **Whisper API**: Para transcripción de mensajes de voz

Arquitectura de la solución

La arquitectura del sistema sigue un patrón de diseño MVC (Modelo-Vista-Controlador) adaptado a las particularidades de un bot de Telegram:

1. **Modelos (Models)**:
    - Definen la estructura de datos (articulo.js, pedido.js, user.js, adminRequest.js)
    - Implementan validaciones y relaciones
    - Proporcionan métodos para interactuar con la base de datos
2. **Vistas (Views)**:
    - En este caso, las "vistas" son los mensajes y botones de Telegram
    - Los servicios de generación de botones actúan como la capa de presentación
3. **Controladores (Controllers)**:
    - Manejan la lógica de negocio y orquestan el flujo de datos
    - Organizados por funcionalidad (cartController, productController, etc.)
    - Modulares y con responsabilidades bien definidas
4. **Servicios (Services)**:
    - Implementan funcionalidades específicas (botStateService, carritoService, etc.)
    - Actúan como capa de abstracción para APIs externas
    - Proporcionan funcionalidades reutilizables
5. **Handlers**:
    - Manejan tipos específicos de interacciones (comandos, callbacks, etc.)
    - Implementan la lógica de enrutamiento de mensajes
    - Gestionan estados de conversación

Diagrama de la arquitectura:

Funcionalidades principales

1. **Gestión de usuarios**:
    - Registro automático de usuarios al interactuar con el bot
    - Sistema de permisos para administradores
    - Solicitud y aprobación de permisos administrativos
2. **Gestión de productos**:
    - Búsqueda de productos mediante texto natural
    - Visualización de detalles
    - Verificación de disponibilidad en tiempo real
    - Importación masiva mediante CSV/Excel
3. **Gestión de carrito**:
    - Añadir productos al carrito
    - Modificar cantidades
    - Eliminar productos
    - Ver resumen del carrito
    - Exportar carrito a JSON/CSV
4. **Procesamiento de pedidos**:
    - Creación de pedidos
    - Verificación de stock
    - Actualización de inventario
    - Seguimiento de estado
5. **Funcionalidades avanzadas**:
    - Procesamiento de mensajes de voz mediante OpenAI Whisper
    - Análisis de intención mediante OpenAI (NLU)
    - Búsqueda semántica mediante embeddings vectoriales
    - Manejo de estados de conversación para experiencia fluida

Decisiones técnicas relevantes

1. **Arquitectura modular**:
    - Se ha optado por una estructura modular con componentes claramente separados

    - Cada funcionalidad está encapsulada en su propio módulo
    - Facilita mantenimiento y escalabilidad
2. **Uso de MongoDB**:
    - Base de datos NoSQL que permite flexibilidad en el esquema
    - Facilita el almacenamiento de embeddings vectoriales
    - Permite escalabilidad horizontal
3. **Implementación de análisis de intención**:
    - Uso de OpenAI para entender lo que el usuario quiere hacer (NLU)
    - Mantenimiento de estado de conversación para contexto
    - Aproximación híbrida con patrones regulares para optimizar costes
4. **Sistema de reintentos**:
    - Implementación de mecanismos de reintento para operaciones críticas
    - Manejo de desconexiones de MongoDB
    - Gestión de errores en llamadas a APIs externas
5. **Organización de controladores por funcionalidad**:
    - Controladores específicos para carrito, productos, checkout, etc.
    - Subdivisión en módulos para mejor mantenimiento
    - Patrón de índice para simplificar importaciones

2.4. Evaluación y Resultados

Objetivos logrados:

- Sistema completo de gestión de inventario
- Carrito de compra funcional
- Búsqueda semántica mediante embeddings
- Panel de administración básico
- Procesamiento de voz y texto
- Gestión de estados de conversación

Ampliaciones realizadas:

- Transcripción de mensajes de voz
- Sistema de permisos para administradores
- Exportación de carrito a diferentes formatos
- Implementación de reintentos para operaciones críticas

Grado de implementación:

- 90% de las funcionalidades planificadas han sido implementadas
- Algunas funcionalidades avanzadas (pasarela de pagos, estadísticas detalladas) quedan para futuras versiones

Herramientas utilizadas:

- GitHub para control de versiones
- Jira para gestión de tareas
- MongoDB Atlas como base de datos de pruebas
- MongoDB como base de datos en producción
- DigitalOcean para despliegue
- OpenAI para procesamiento de lenguaje natural (NLU)

Incidencias y resoluciones

Problemas encontrados:

1. **Gestión de contexto en conversaciones**:
    - *Problema*: Dificultad para mantener el contexto entre mensajes del usuario
    - *Solución*: Implementación de un sistema de estados y contexto por usuario
2. **Precisión en análisis de intención**:
    - *Problema*: Falsos positivos en la detección de intenciones
    - *Solución*: Refinamiento de prompts y combinación con patrones regulares
3. **Consumo de API de OpenAI**:
    - *Problema*: Costes elevados en uso intensivo
    - *Solución*: Implementación de caché y optimización de llamadas
4. **Manejo de archivos de audio**:
    - *Problema*: Dificultades con la API de Telegram para descargar archivos
    - *Solución*: Implementación de sistema propio de descarga con reintentos
5. **Concurrencia en actualizaciones de inventario**:
    - *Problema*: Posibles condiciones de carrera al actualizar stock
    - *Solución*: Uso de operaciones atómicas de MongoDB

Lecciones aprendidas:

1. **Importancia del logging**:
    - Un sistema robusto de logging facilita enormemente la depuración
    - La implementación temprana de Winston fue de gran ayuda
2. **Modularidad como clave del éxito**:
    - La estructura modular facilitó el desarrollo paralelo
    - Permitió refactorizar componentes sin afectar al resto
3. **Pruebas continuas**:
    - Es esencial probar cada funcionalidad en condiciones reales
    - El testing manual constante evitó problemas mayores
4. **Gestión de dependencias externas**:
    - Es crucial implementar mecanismos de reintento para APIs externas
    - La monitorización de cuotas y límites debe ser proactiva
5. **Importancia de la experiencia de usuario**:
    - Un bot conversacional debe ser natural y fácil de usar
    - El análisis de intención mejora significativamente la experiencia

3. Elementos Finales

3.1. Conclusiones

Resumen de los resultados

El proyecto ha culminado con éxito en el desarrollo de un sistema de comercio electrónico usando Telegram, que permite a pequeños negocios gestionar su inventario y ventas de manera eficiente. El sistema implementa tecnologías avanzadas como análisis de intención mediante IA y búsqueda semántica, proporcionando una experiencia de usuario natural y fluida.

Los principales logros incluyen:

- Sistema completo de gestión de inventario con importación desde CSV/Excel
- Carrito de compra funcional con verificación de stock en tiempo real
- Búsqueda inteligente de productos mediante embeddings vectoriales
- Panel de administración integrado en el mismo bot
- Procesamiento de mensajes de voz mediante OpenAI Whisper
- Arquitectura modular y escalable para futuras ampliaciones

Valoración personal

El desarrollo de este proyecto ha sido un desafío técnico significativo que ha permitido explorar tecnologías emergentes como el procesamiento de lenguaje natural y los embeddings vectoriales. La implementación de un sistema conversacional ha requerido un enfoque diferente al desarrollo web tradicional, centrándonos en la experiencia del usuario y en cómo hacer que la interacción sea lo más natural posible.

La arquitectura modular adoptada ha demostrado ser una decisión acertada, facilitando el desarrollo paralelo y el mantenimiento del código. La integración con servicios externos como OpenAI ha añadido capacidades avanzadas al sistema, aunque también ha presentado desafíos en términos de gestión de costes y dependencias.

En general, consideramos que el proyecto ha cumplido sus objetivos principales y ha resultado en un producto funcional y con potencial para ser utilizado en entornos reales.

Propuestas de mejora

A pesar del éxito del proyecto, identificamos varias áreas de mejora para futuras versiones:

1. **Optimización de costes de API**:
    - Implementación de un sistema de caché más sofisticado
    - Explorar alternativas locales para embeddings (como sentence-transformers)
2. **Mejoras en la interfaz**:
    - Añadir más elementos visuales (imágenes de productos)
    - Implementar mensajes con formato enriquecido
3. **Funcionalidades adicionales**:
    - Integración con pasarelas de pago
    - Sistema de notificaciones para administradores
    - Análisis de datos y estadísticas avanzadas
4. **Mejoras técnicas**:
    - Implementación de pruebas automatizadas
    - Optimización de rendimiento en grandes volúmenes de datos
    - Mejora en la gestión de concurrencia

Líneas futuras de desarrollo

Para la evolución del proyecto, se identifican las siguientes líneas de desarrollo:

1. **Panel web administrativo**:
    - Desarrollo de una interfaz web para administradores
    - Visualización de datos y estadísticas
    - Herramientas avanzadas de gestión de inventario
2. **Integración con sistemas externos**:
    - Conexión con ERPs y sistemas de gestión
    - Integración con marketplaces
    - Sincronización con tiendas físicas
3. **Expansión a otras plataformas**:
    - Adaptación para WhatsApp Business
    - Implementación para Facebook Messenger
    - Desarrollo de chatbot web
4. **Personalización y marketing**:
    - Sistema de recomendaciones personalizado
    - Campañas automatizadas
    - Segmentación de clientes

3.2. Referencias y Bibliografía

Referencias técnicas

- **Node.js Documentation**: https://nodejs.org/docs/
- **MongoDB Documentation**: https://docs.mongodb.com/
- **Telegram Bot API**: https://core.telegram.org/bots/api
- **OpenAI API Reference**: https://platform.openai.com/docs/api-reference
- **Express.js Documentation**: https://expressjs.com/en/guide/routing.html

Bibliografía consultada

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*.
- Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*.

Recursos web

- **OpenAI Documentation**: https://platform.openai.com/docs/
- **Mongoose Documentation**: https://mongoosejs.com/docs/

- **Node-Telegram-Bot-API GitHub**: https://github.com/yagop/node-telegram-bot-api
- **Embedding bases**:

  https://www.intersystems.com/es/recursos/que-son-los-vector-embeddings/

- **Embeddings multidimencionalidad**: https://simonwillison.net/2023/Oct/23/embeddings/
- **Embeddings ejemplos**:

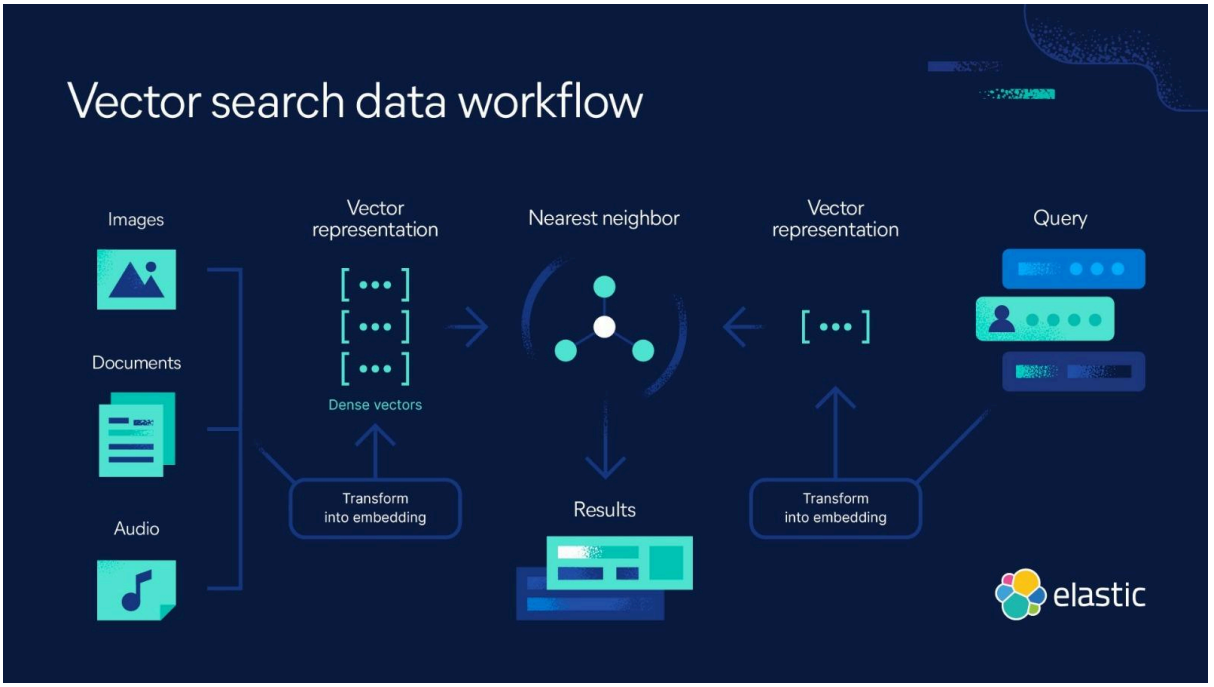
  https://www.youtube.com/watch?v=Tg1MjMIVArc

Herramientas utilizadas

- **Visual Studio Code**: Entorno de desarrollo
- **Git/GitHub**: Control de versiones
- **Jira**: Gestión de tareas y seguimiento de proyecto
- **Postman**: Testing de API
- **MongoDB Compass**: Visualización y gestión de base de datos
- **DigitalOcean**: Plataforma de despliegue
- **BotFather (Telegram)**: Creación y configuración del bot

3.3. Anexos

- Capturas de pantalla del bot en funcionamiento
- Diagramas detallados de la arquitectura
    - Estructura de los embeddings:



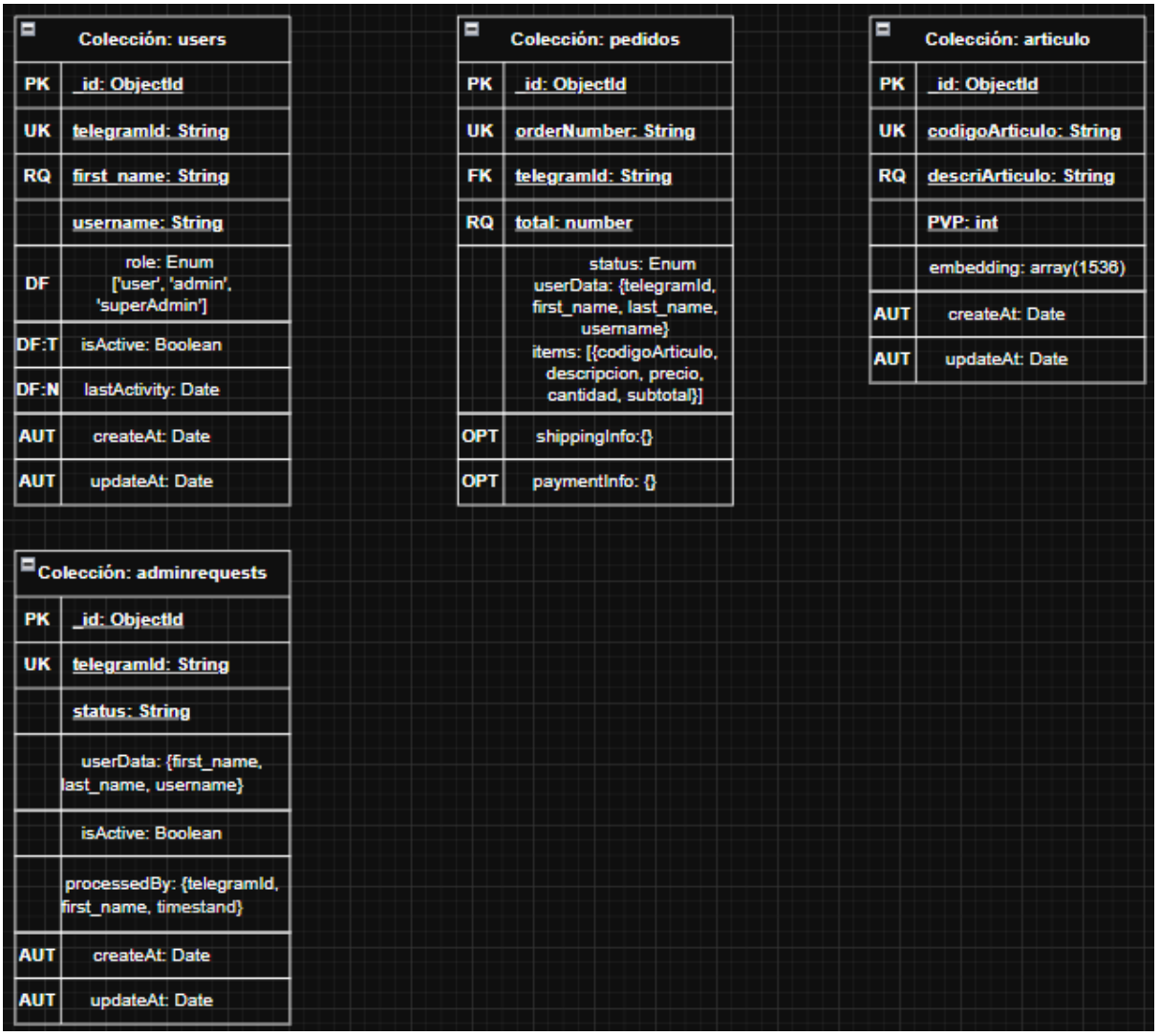
- Diagrama del backend:

```
WHAISPER/
├── /config
│ ├── constants.js # Constantes globales y estados de conversación
│ └── database.js # Conexión MongoDB con reintentos automáticos
├── /controllers
│ ├── /admin
│ │ ├── managementController.js # Gestión de administradores
│ │ └── requestsController.js # Solicitudes para ser administrador
│ ├── /cart
│ │ ├── addModule.js # Añadir productos al carrito
│ │ ├── displayModule.js # Mostrar carrito
│ │ ├── exportModule.js # Exportar carrito (JSON/CSV)
│ │ ├── index.js # Unifica funcionalidades del carrito
│ │ ├── removeModule.js # Eliminar productos del carrito
│ │ └── updateModule.js # Actualizar cantidades del carrito
│ ├── /product
│ │ ├── index.js # Unifica funcionalidades de productos
│ │ ├── quantityModule.js # Selección de cantidad con validación
de stock
│ │ ├── searchModule.js # Búsqueda de productos con IA
│ │ └── selectModule.js # Selección de productos
│ ├── adminController.js # Control de acceso y funciones
administrativas
│ ├── aiController.js # Búsqueda por similitud con embeddings
│ ├── audioController.js # Procesamiento de mensajes de voz/audio
│ ├── botController.js # Controlador principal del bot
│ ├── cartController.js # Interfaz principal para gestión del
carrito
```

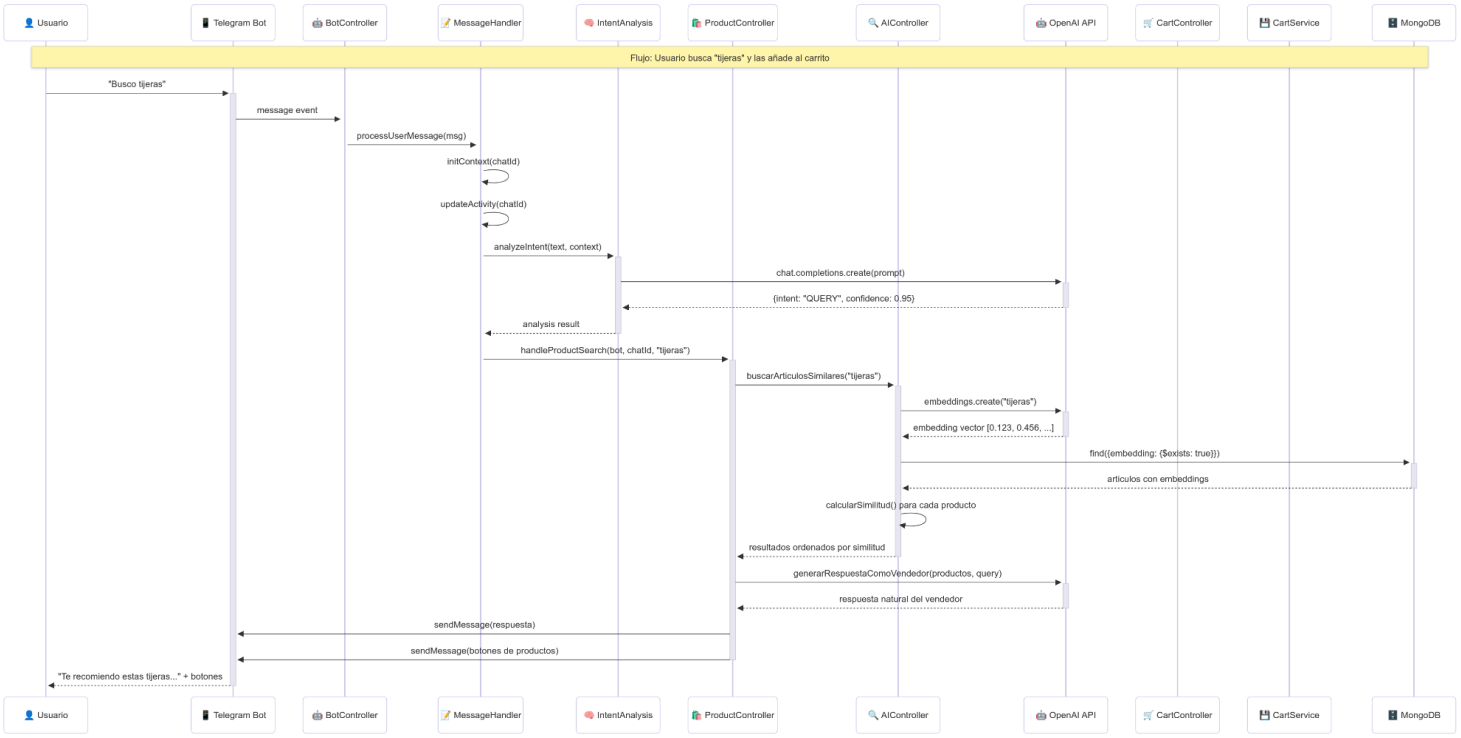
```
| ├── checkoutController.js # Proceso de tramitación de pedidos
| ├── conversationController.js # Gestión del flujo de conversación | ├── productController.js # Interfaz principal para productos
| └── uploadController.js # Subida y procesamiento de archivos
├── /handlers
| ├── callbackHandlers.js # Manejo de callbacks de botones
| ├── commandHandlers.js # Comandos del bot (/admin, /carrito, etc.) | ├── errorHandlers.js # Manejo global de errores
| ├── index.js # Exporta todos los handlers
| ├── intentHandlers.js # Manejo basado en intenciones detectadas
| ├── messageHandlers.js # Procesamiento principal de mensajes
| └── stateHandlers.js # Manejo de estados de conversación
├── /logs
| ├── combined.log # Logs combinados
| └── error.log # Solo errores
├── /middleware
| ├── auth.js # Autenticación JWT y verificación de roles
| └── uploadMiddleware.js # Configuración de subida de archivos
├── /models
| ├── adminRequest.js # Solicitudes de permisos de administrador
| ├── articulo.js # Productos/artículos del inventario
| ├── pedido.js # Pedidos realizados
| └── user.js # Usuarios del sistema
├── /routes
| └── api.js # Rutas de la API REST
├── /services
| ├── adminService.js # Gestión centralizada de administradores
| ├── botStateService.js # Estados de conversación y contexto
```

```
| ├── buttonGeneratorService.js # Generación de botones interactivos | ├── carritoService.js # Carrito de compras en memoria
| ├── fileProcessingService.js # Procesamiento de CSV/Excel
| ├── generarRespuestaComoVendedor.js # Respuestas naturales con IA | ├── intentAnalysisService.js # Análisis de intenciones con OpenAI | ├── inventoryService.js # Gestión de inventario y stock
| ├── openaiService.js # Servicios de OpenAI (embeddings)
| ├── orderService.js # Gestión de pedidos
| ├── telegramService.js # Instancia del bot de Telegram
| └── whaisperService.js # Transcripción de audio con Whisper
├── /utils
| ├── helpers.js # Cálculo de similitud coseno
| ├── logger.js # Sistema de logging
| ├── telegramDownloader.js # Descarga de archivos de Telegram
| └── telegramUtils.js # Utilidades para envío con reintentos
├── /temp # Archivos temporales para audio/exports
├── /uploads # Archivos subidos por administradores
├── generateEmbeddings.js # Script para generar embeddings
└── index.js # Punto de entrada principal
```

- Esquema de la BD:


Colección: users
PK _id: ObjectId
UK telegramId: String
RQ first_name: String
username: String
DF role: Enum ['user', 'admin', 'superAdmin']
DF:T isActive: Boolean
DF:N lastActivity: Date
AUT createAt: Date
AUT updateAt: Date
Colección: pedidos
PK _id: ObjectId
UK orderNumber: String
FK telegramId: String
RQ total: number
status: Enum
userData: {telegramId, first_name, last_name, username}
items: [{codigoArticulo, descripcion, precio, cantidad, subtotal}]
OPT shippingInfo:{}
OPT paymentInfo: {}
Colección: articulo
PK _id: ObjectId
UK codigoArticulo: String
RQ descriArticulo: String
PVP: int
embedding: array(1536)
AUT createAt: Date
AUT updateAt: Date
Colección: adminrequests
PK _id: ObjectId
UK telegramId: String
status: String
userData: {first_name, last_name, username}
isActive: Boolean
processedBy: {telegramId, first_name, timestand}
AUT createAt: Date
AUT updateAt: Date

- Esquema UML:



- Ejemplos de código relevantes
- Logs de rendimiento

```
[LOG]: Iniciando bot de Telegram...
[LOG]: Comandos del bot registrados
[LOG]: Manejadores de errores globales configurados
[LOG]: Bot configurado y listo para recibir mensajes
(node:6779) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0
and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:6779) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version
4.0.0 and will be removed in the next major version
(node:6779) [MONGOOSE] Warning: Duplicate schema index on {"CodigoArticulo":1} found. This is often due to declaring an index using both "index
: true" and "schema.index()". Please remove the duplicate index definition.
🚀 Servidor web en http://localhost:3000
[LOG]: ✅ MongoDB conectado correctamente
[LOG]: Usuario 667170889: "Hola quiero algo de beber"
[LOG]: Estado actual: initial
[LOG]: Intención detectada: QUERY (0.9)
[LOG]: Usuario 667170889: "Quiero agua"
[LOG]: Estado actual: showing_products
[LOG]: Intención detectada: QUERY (0.9)
[LOG]: Callback recibido: reject_products de usuario 667170889
[LOG]: Usuario 667170889: "Quiero agua"
[LOG]: Estado actual: initial
[LOG]: Intención detectada: QUERY (0.95)
[LOG]: Usuario 667170889: "Tienes algo para calcular?"
[LOG]: Estado actual: showing_products
[LOG]: Intención detectada: QUERY (0.9)
[LOG]: Callback recibido: reject_products de usuario 667170889
[LOG]: Callback recibido: reject_products de usuario 667170889
```

- Estadísticas de precio por token: